Database Systems
CSE 344

Lectures 9: Relational Algebra (part 2) and Query Evaluation (Ch. 5.2 & 16.3 (skim 16.3.2))
Announcements

• WQ3 is due on Monday July 10
• HW1 grades were be posted
• HW3 - Get Azure Setup
You will get an email like this to your Azure account.
Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$ (Sigma)
- Projection $\pi$ ($\Pi$) (Pi)
- Cartesian product $\times$, join $\bowtie$
- Rename $\rho$ (Rho)
- Duplicate elimination $\delta$ (Delta)
- Grouping and aggregation $\gamma$ (Gamma)
- Sorting $\tau$ (Tau)

Extended RA
Join Summary

• **Theta-join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  – Join of $R$ and $S$ with a join condition $\theta$
  – Cross-product followed by selection $\theta$

• **Equijoin**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  – Join condition $\theta$ consists only of equalities

• **Natural join**: $R \bowtie S = \pi_A (\sigma_{\theta}(R \times S))$
  – Equijoin
  – Equality on all fields with same name in $R$ and in $S$
  – Projection $\pi_A$ drops all redundant attributes
More Joins

• Outer join
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
  – Does not eliminate duplicate columns

• Variants
  – Left outer join (⪫)
  – Right outer join (⪬)
  – Full outer join (⪯)
More Examples

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)

Name of supplier of parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part})) \]

Name of supplier of red parts or parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}=\text{'red'}} (\text{Part}) ) \]

\[ \bowtie \simeq \cup \]

\[ \sigma_{\text{psize}>10} \lor \sigma_{\text{pcolor}=\text{'red'}} \]
Query Evaluation Steps

1. Parse & Check Query
   - Translate query string into internal representation
   - Check syntax, access control, table names, etc.
   - Logical plan → physical plan

2. Decide how best to answer query:
   - Query optimization

3. Query Execution

4. Return Results
From SQL to RA

• SQL → RA → Syntax Tree
• Graphical representation of evaluation order.

```
  +
 /   \
+     5
|     |
3     4

(3 + 4) * 5
```

```
  +
 /   \
3     *
|     |
4     5

3 + 4 * 5
```
From SQL to RA

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and x.price > 100 and z.city = 'Seattle'

Is there a better plan?
RA: Equivalence Transformations

Conjunctive selection operations can be deconstructed

$$\sigma_{\theta_1 \land \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

Selection operations are commutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_2}(E))$$

Theta joins are commutative

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

Natural joins are associative

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
      x.price > 100 and
      z.city = 'Seattle'

Can you think of another plan?
From SQL to RA

```
SELECT DISTINCT x.name, z.name 
FROM Product x, Purchase y, Customer z 
WHERE x.pid = y.pid and y.cid = z.cid and 
  x.price > 100 and 
  z.city = 'Seattle'
```

Can you think of another plan?

Query optimization: find an equivalent optimal plan

Push selections down the query plan!

CSE 344 – Summer 2017
Extended RA: Operators on Bags

• Duplicate elimination $\delta$

• Grouping $\gamma$
  – Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.

• Sorting $\tau$
  – Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.
Using Extended RA Operators

```sql
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

T1, T2, T3 = temporary tables

T1(city, p, c)
T2(city, p, c)
T3(city, c)

\[
\begin{align*}
T1 & \xrightarrow{\gamma} T2 & \xrightarrow{\sigma} T3
\end{align*}
\]

sales(product, city, price)
Typical Plan for a Query (1/2)

Answer

π_{fields}

σ_{selection condition}

join condition

join condition

R

S

SELECT fields
FROM R, S, ...
WHERE condition

SELECT-PROJECT-JOIN
Query
Typical Plan for a Query (1/2)

\[
\begin{align*}
\pi_{\text{fields}} & \quad \sigma_{\text{having condition}} \\
\gamma_{\text{fields, sum/count/min/max(fields)}} & \quad \sigma_{\text{where condition}} \\
& \quad \text{join condition} \\
& \quad \ldots \\
& \quad \ldots
\end{align*}
\]

SELECT fields
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition
How about Subqueries?

```sql
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
and not exists
  (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
     and P.price > 100)
```

Correlation!
How about Subqueries?

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
       and P.price > 100)
```

De-Correlation

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
    and Q.sno not in
      (SELECT P.sno
       FROM Supply P
       WHERE P.price > 100)
```
How about Subqueries?

\[
\begin{align*}
\text{(SELECT } & Q.\text{sno} \\
\text{FROM } & \text{Supplier Q} \\
\text{WHERE } & Q.\text{sstate} = 'WA') \\
\text{EXCEPT} & \\
\text{(SELECT } & P.\text{sno} \\
\text{FROM } & \text{Supply P} \\
\text{WHERE } & P.\text{price} > 100) \\
\end{align*}
\]

\text{EXCEPT} = \text{set difference}
How about Subqueries?

\[
\pi_{\text{sno}} (\sigma_{\text{sstate}='\text{WA}'} (Q.\text{sno} \\
\text{FROM Supplier } Q \\
\text{WHERE } Q.\text{sstate} = '\text{WA}')) \\
\text{EXCEPT} \\
(\sigma_{\text{Price} > 100} (P.\text{sno} \\
\text{FROM Supply } P \\
\text{WHERE } P.\text{price} > 100))
\]
From Logical Plans to Physical Plans
Relational Algebra

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'

Give a relational algebra expression for this query
\[ \pi_{\text{sn}} \left( \sigma_{\text{scity='Seattle' \land sstate='WA' \land pno=2}} \left( \text{Supplier} \bowtie_{\text{sid=sid}} \text{Supply} \right) \right) \]
Relational Algebra

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

Relational algebra expression is also called the “logical query plan”
Query Evaluation Steps Review

1. **Parse & Rewrite Query**
2. **Select Logical Plan**
3. **Select Physical Plan**
4. **Query Execution**

**Disk**

**Query optimization**

**Logical plan (RA)**

**Physical plan**
Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join
A physical query plan is a logical query plan annotated with physical implementation details.

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'
Main Memory Algorithms

Logical operator:

\[ \text{Product}(\text{pid, name, price}) \bowtie_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid, cid, store}) \]

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join \( O(n^2) \)
2. Merge join \( O(n \log n) \)
3. Hash join \( O(n) \ldots O(n^2) \)

add n to hash – \( O(n) \)?
lookup n in hash – \( O(n) \)?
Physical Query Plan 2

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Supplier\((\text{sid}, \text{sname}, \text{scity}, \text{sstate})\)
Supply\((\text{sid}, \text{pno}, \text{quantity})\)

**Physical Query Plan 3**

\[
\pi_{\text{sname}} \quad \sigma_{\text{scity} = \text{Seattle} \land \text{sstate} = \text{WA}} \\
\]

(On the fly)

(Sort-merge join)

(Scan & write to T1)

\[
\sigma_{\text{scity} = \text{Seattle} \land \text{sstate} = \text{WA}} \\
\]

(Sort-merge join)

(Scan & write to T2)

\[
\sigma_{\text{pno} = 2} \\
\]

Different but equivalent logical query plan; different physical plan

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{‘Seattle’} \\
& \quad \text{and} \ x.\text{sstate} = \text{‘WA’}
\end{align*}
\]
Query Optimization Problem

- For each SQL query... many logical plans

- For each logical plan... many physical plans

- How do find a fast physical plan?
  - Will discuss in a few lectures
  - First we need to understand how query operators are implemented
Query Execution
Iterator Interface for Query Operators (Relations)

- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition

- **next()**
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple

- **close()**: clean-up state
Pipelined Query Execution

(On the fly)

σ_{scity='Seattle' and sstate='WA' and pno=2}

(On the fly)

π_{sname}

(Nested loop)

sno = sno

Suppliers
(File scan)
(open())

Supplies
(File scan)
(open())

(open())
Pipelined Query Execution

(On the fly) \( \pi_{\text{sname}} \)

(On the fly) \( \sigma_{\text{scity}=\text{Seattle} \text{ and sstate}=\text{WA} \text{ and pno}=2} \)

(Nested loop) \( \text{sno} = \text{sno} \)

Suppliers
(File scan)

Supplies
(File scan)

In 444 you implement this.
Pipelined Execution

• Tuples generated by an operator are immediately sent to the parent

• Benefits:
  – No operator synchronization issues
  – No need to buffer tuples between operators
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk

• This approach is used whenever possible
Query Execution Bottom Line

- SQL query transformed into **physical plan**
  - **Access path selection** for each relation
    - Scan the relation or use an index (next lecture)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization

- Pipelined execution of physical plan
Physical Data Independence

• Applications are insulated from changes in physical storage details

• SQL and relational algebra facilitate physical data independence
  – Both languages input and output relations
  – Can choose different implementations for operators